PATENT
5854-00200

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/700,152
Confirmation No. 3829

I hereby certify that this correspondence is being transmitted to the United States Patent & Trademark Office via electronic submission or facsimile on the date indicated below:

12/08/2008 /Pamela Gerik/
Date Pamela Gerik

APPEAL BRIEF

Dear Sir:

Further to the Notice of Appeal filed July 8, 2008, Appellant presents this Appeal Brief. Appellant hereby appeals to the Board of Patent Appeals and Interferences from the rejection of pending claims 1-20 and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by Ramal Acquisition Corp., a wholly owned subsidiary of Pervasive Software, Inc., as evidenced by the document recorded at reel 018779 and frame 0519.

II. RELATED APPEALS AND INTERFERENCES

No appeals, interferences, or judicial proceedings are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF THE CLAIMS

Claims 1 -20 are pending in the application and stand rejected.

IV. STATUS OF AMENDMENTS

No claim amendments were filed subsequent to their final rejection. (Applicants reserve the right to continue prosecution after the Board's decision and specifically notes several typographical errors.) The Appendix hereto reflects the current state of the claims.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Independent claim 1 recites a method for developing a dataflow application comprising: developing one or more data transformations using a host language **(Specification -- pg. 7, lines 15-19; pg. 17, lines 8-13; pg. 16, lines 18-20; pg. 21, lines 20-22)**; assembling several data transformations having ports into a map component with links between ports using a declarative language for static assemblage and a host language for dynamic assemblage, wherein a map component performs a respective data transformation **(Specification -- pg. 7, lines 15-18; pg. 8, lines 1-4; pg. 26, lines 1-12)**; compiling one or more map components with syntactic and semantic analysis **(Specification -- pg. 7, lines 20-22; pg. 15, lines 12-14; pg. 16, lines 26-28; pg. 26, lines 1-12)**; and synthesizing the compiled map components into an executable dataflow application including removing the design time links between ports **(Specification -- pg. 7, lines 20-23; pg. 15, lines 1-19; pg. 24, lines 10-14; Fig. 10)**.

Independent claim 14 recites a computer-readable medium having a dataflow development system comprising: a library of components, some of the components being scalar and comprising a data transformation, some of the components being composite, and comprising a hierarchy of other components representing data transformations **(Specification -- pg. 7, lines 8-10; pg. 11, lines 23-26; pg. 21, lines 24-31; Fig. 1)**; a compiler to develop and assemble components into an executable dataflow application linking components subject to schema and properties constraints **(Specification --pg. 11, lines 15-16; pg. 26, lines 1-5; pg. 43, lines 15-18;**

Fig. 1); an executor which executes the dataflow application in parallel **(Specification -- pg. 11, lines 20-26; pg. 16, lines 14-23; pg. 21, lines 24-26)**; and tools for accessing the functionality of the compiler, executor and component library **(Specification -- pg. 11, lines 20-26; pg. 16, lines 1-5; pg. 23, lines 18-22; Fig. 1)**.

Independent claim 17 recites a computer-readable medium having a dataflow application development environment where map components are selected from a plurality of reusable map components each representing one or more data transformations **(Specification -- pg. 7, lines 8-10; pg. 11, lines 23-26; pg. 21, lines 24-31; Fig. 1)**, the selected map components are visually assembled into a dataflow application, patterns of parallelism recognized **(Specification -- pg. 8, lines 3-5; pg. 19, lines 20-25**, and at least some of the components executed in parallel by assigning a number of threads to a respective number of data transformations **(Specification -- pg. 8, lines 3-5; pg. 29, lines 18-22; pg. 30, lines 6-8**, and at least some of the components are scalar components with each scalar component executed on a separate thread **(Specification -- pg. 20, lines 6-11; Fig. 25)**.

Data processing in a business process takes a variety of forms, such as data warehousing, decision support software, analytical software, and customer relationship management. Such data processing invariably involves transforming the data for use. Dataflow graphs are widely recognized as an effective means to specify data processing software. Their value lies in the succinct presentation and explicit definition of data transfers as the data progresses through a chain of transformation processes.

The present invention relates to computationally efficient data transformation applications using dataflow graphs particularly suited for parallel architectures when the synthesis process is decomposed into a sequence of steps. The claimed system and methods center on the decomposition of the synthesis task into a sequence of steps such that a dataflow application is not generated immediately from a map component which has been created by the application designer. Rather, the designer works with a map component editor which manipulates iconic objects by using graphical and text editors and importing map components from libraries of existing maps.

Broadly claim 1 contemplates a method for developing a dataflow application where one or more data transformations are developed using a host language and several data transformations having ports are assembled into a map component with links between ports using a declarative language for static assemblage and a host language for dynamic assemblage. One or more map components are compiled with syntactic and semantic analysis and the compiled map components are synthesized into an executable dataflow application including removing the design time links between ports.

No means plus function or step plus function type claims are presented.

VI. GROUNDS OF REJECTION TO BE REVIEWED

1. Claims 14-20 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter.

2. Claims 1, 2, 4-14, and 16 stand rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,874,141 to Swamy et al. (hereinafter "Swamy").

3. Claims 3 and 15 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Swamy in view of U.S. Patent No. 6,947,947 Block et al. (hereinafter "Block").

4. Claims 17-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Swamy in view of U.S. Patent No. 6,449,619 Colliat et al. (hereinafter "Colliat").

VII. ARGUMENT

1. The rejection of claims 14-20 under 35 U.S.C. § 101

The final Office Action rejected claims 14-20 under 35 USC 101 as directed to non-statutory subject matter. The final Office Action (page 5) notes that claims 14-20 recite a computer readable medium having a series of elements interpreted as software. However, the

Final Office Action objects to the transitional phrase "having" because it does not necessarily imply storage of an executable program capable of causing a computer to carry out the recited elements.

While "storing" might be a better transitional term in claim 14, the term "having" has been used as a transitional phrase and interpreted as reciting an executable program for causing the recited functionality. *Ex parte Bo Li*, Appeal 2008-1213 (BPAI 2008) considered a similar claim. In citing the new test under *In re Biliski*, the Board overturned the § 101 rejection finding that the product claims cites statutory subject matter. The claim at issue in *In re Biliski* stated as follows:

> 42. A computer program product, comprising a computer usable medium **having** a computer readable program code embodied therein, said computer readable program code adapted to be executed to implement a method for generating a report . . . (emphasis added).

The Board noted:

> It has been the practice for a number of years that a "Beauregard Claim" of this nature be considered statutory at the USPTO as a product claim. (MPEP 2105.01, I). Though not finally adjudicated, this practice is not inconsistent with *In re Nuijten*. Further, the instant claim presents a number of software components, such as the claimed logic processing module, configuration file processing module, data organization module, and data display organization module, that are embodied upon a computer readable medium. This combination has been found statutory under the teachings of *In re Lowry*, 32 F.3d 1579 (Fed. Cir. 1994). In view of the totality of these precedents, we decline to support the rejection under 35 U.S.C. § 101.

In the present case, claim 14 recites "[a] computer-readable medium **having** a dataflow development system" (emphasis added) and then recites a series of software elements embodied upon the computer-readable medium. Applicant believes that the use of the term "storing" versus "having" is not necessary, as long as a person of ordinary skill in the art would understand that the recited elements are embodied on the medium – clearly the case here. Claim 17 presents the identical issue and is similarly believed to claim statutory subject matter.

For at least the aforementioned reasons, Applicants believe claims 14-20 are directed to statutory subject matter.

2. Rejection of claims 1, 2, 4-14, and 16 under 35 U.S.C. § 102(e)

Claims 1, 2, 4-14, and 16 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Swamy. The standard for "anticipation" is one of fairly strict identity. A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art of reference. *Verdegaal Bros. v. Union Oil Co. of California*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987); MPEP 2131. Furthermore, anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, as arranged in the claim. *W.L. Gore & Assocs. V. Garlock*, 721 F.2d 1540, 220 USPQ 303 (Fed. Cir. 1983); *Net Moneyin. v. Verisign* (Fed. Cir. 2008).

The diagrams used in Swamy show the relationships between the source and target schema for the maps. The Swamy diagram is declarative and requires a compiler to generate code for an execution engine to perform the data transformation. In the present application, components are assembled into applications or higher order components. The compiler is responsible for ensuring that components are linked correctly (port data type checking and synthesis) and to resolve references to composite components. The executor is responsible for initializing and running the components, and for conveying data across the links. The dataflow graph is actually executed. In the present application, the components perform the data transformations – in contrast to Swamy.

In Swamy, links are used to relate nodes in the source and target trees to one another or to the ports on functoids. In Swamy, a functoid appears to be a component that operates one or more scalar components to produce a single scalar output. The links may imply flow of data; however, the actual movement of data is determined by analysis of the source and target schemas and is performed by the engine. However, in the present invention, the diagrams are dataflow graphs where the links represent components that convey data from output port to input port. In

fact, the links, ports, and map components are all real components in the framework of the present invention.

The compilation operation in Swamy uses the mapping to generate code in a host language that can be executed by an engine or processor to perform the translation or transformation of data in the source tree structure to data that conforms with the target tree structure. In the present invention, the compilation process prepares assemblies of components for execution. This process involves the loading and expansion of static subassemblies (composite components), configuration of components, the type synthesis of generic ports, and the optimization of the dataflow graph. There is no generation of code in a different host language. Swamy does not generate an executable dataflow application as called for in present claims 1 and 14.

In the system described by Swamy, the mapping is translated into a host language. Swamy is, in fact, a system for code generation. In the exemplary implementation the target language for the code generator is XSLT. However, the present application describes a framework for assembling components into applications or higher order components. The foundation components and scalar map components, are written by users in a host language and conform to interfaces defined in the framework.

Swamy does not perform port data type synthesis. In fact, it would appear that the Swamy invention treats all data as strings of text. In addition, the compilation process used in the Swamy invention does not produce an executable dataflow graph. The result in the preferred implementation is an XSLT script. That is, Swamy does not create an executable dataflow application as called for by claims 1 and 14.

Recently, in *Net MoneyIn v. Verisign* (Fed. Cir. 2008) the Federal Circuit affirmed that anticipation takes *more* than simply locating each element within the four corners of a single document. To anticipate, the prior art must teach all the claim elements and the claimed arrangement. Because the hallmark of anticipation is prior invention, the prior art reference – in order to anticipate under 35 U.S.C. § 102 – must not only disclose all elements of the claim

within the four corners of the document, but must also disclose those elements "arranged as in the claim."

As noted above, while Swamy and the present claims may contain similar key words, Swamy does not teach all of the elements of claims 1, 14, and 17, nor the arrangement of the claim elements.

a) Swamy does not perform port data type synthesis. In fact, it would appear that the Swamy invention treats all data as strings of text. See, e.g. claim 1 "wherein a map component performs a respective data transformation" and "compiling one or more map components with syntactic and semantic analysis . . ."

b) The compilation process used in Swamy does not produce an executable dataflow graph. That is, Swamy does not create an executable dataflow application. Claim 1 calls for "synthesizing the compiled map components into an executable dataflow application." Claim 14 calls for a "compiler to develop and assemble components into an executable dataflow application . . ."

c) In the system described by Swamy, the mapping is translated into a host language – it is in fact a system for code generation. However, the present application describes and claims a framework for assembling components into applications or higher order components. The foundation components and scalar map components, are written by users in a host language and conform to interfaces defined in the framework. For example, claim 1 calls for "developing one or more data transformations using a host language" and "assembling several data transformations having ports into a map component with links between ports using a declarative language for static assemblage and a host language for dynamic assemblage . . ."

d) In Swamy, links are used to relate nodes in the source and target trees to one another or to the ports on functoids. In Swamy, a functoid appears to be a component that operates one or more scalar components to produce a single scalar output. The links may imply flow of data; however, the actual movement of data is determined by analysis of the source and

target schemas and is performed by the engine. However, in the present invention, the diagrams are dataflow graphs where the links represent components that convey data from output port to input port. In fact, the links, ports, and map components are all real components in the framework of the present invention. *See* claim 1, "synthesizing the compiled map components into an executable dataflow application including removing the design time links between ports."

For at least the aforementioned reasons, Applicants believe claims 1, 2, 4-14, and 16 are not anticipated by Swamy.

3. Rejection of Claims 3 and 15 under 35 U.S.C. § 103(a)

Claims 3 and 15 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Swamy in view of Block. The factual inquiries set forth in *Graham v. John Deere Co.,* 383 U.S. 1, 148 U.S.P.Q. 459 (1966), establish the background for determining obviousness under 35 U.S.C. §103, see also, MPEP 706 and 2141. The primary considerations are: the scope and content of the prior art; the differences between the prior art and the claims in issue; the level of ordinary skill in the pertinent art; and secondary considerations, such as commercial success, long felt and unresolved needs, failures of others, teaching away, etc. See also, *KSR Int'l. Co., v. Teleflex, Inc.,* No. 04-1350 (U.S. Apr. 30, 2007). While the *KSR* court rejected the rigid application of the "teaching, suggestion, motivation" test, the reason a person of ordinary skill in the relevant field would combine prior elements must be made explicit. *KSR*, slip op. at 14.

In the present invention the components are designed to be reusable. Dependent claims 3 and 15 add support for encrypted components, so that users can develop and redistribute interesting component assemblies without fear of disclosing the implementation of the component assembly. Specifically, claims 3 and 15 of the present application relate to encrypting components, such as "encrypting the dataflow graphs" claim 3 and "some of the components being encrypted and comprising a proprietary data transformation" claim 15. As such, in the present invention as claimed the encryption is to protect the intellectual property in the component design and not for working with encrypted inputs and outputs of the component.

In contrast, Block mentions transformations that encrypt or decrypt data.

> The provider and receiver machines can communicate via the Internet. For example, the provider machine can interface the Internet or function as a web server, and the receiver machine can interface the Internet or function as a web browser. Also, the intermediate format can be encrypted, and can be decrypted at the receiver machine in a fashion transparent to a user of the receiver machine. For example, the encryption/decryption mechanism can be a proprietary function of the transformation programs (Block -- col. 7, lines 5-13).

For the reasons stated above referencing Swamy, there is no suggestion of the encryption of map components as called for in claims 3 and 15 by the data encryption techniques of Block. Therefore, applying the tests of Graham and KSR, even if Swamy is combined with Block, there are substantial differences from the present claims and the proposed combination.

For at least the aforementioned reasons, Applicants believe claims 3 and 15 are patentably distinct over the combination of Swamy and Block.

4. <u>The rejection of claims 17-20 under 35 U.S.C. § 103(a)</u>

Claims 17-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Swamy in view of Colliat. In rejecting claims 17-20, the Office Action points to Swamy at Figs. 11-14 (illustrating example mappings with hierarchical components), the selected map components being visually assembled into a dataflow application (see, e.g., Figs. 11-14; col.6, lines 34-39). The diagrams cited in Swamy show the relationships between the source and target schema for the maps. The diagram is declarative and requires a compiler to generate code for an execution engine to perform the data transformation.

In contrast, in the present invention the components are assembled into applications of higher order components. The compiler is responsible for ensuring that components are linked correctly (port data type checking and synthesis) and to resolve references to composite components. The executor is responsible for initializing and running the components and for conveying data across the links. The components perform the data transformations. *See, e.g.,*

claim 17 which recites "some of the components executed in parallel by assigning a number of threads to a respective number of data transformation."

Thus, the diagram styles are different between Swamy and the present invention. In Swamy, the diagram uses links to relate nodes in the source and target trees to one another or to the ports on functoids. The links may imply flow of data; however, the actual movement of data is determined by analysis of the source and target schemas and is performed by the engine. In the present invention, the diagrams are dataflow graphs where the links represent components that convey data from output port to input port. In fact, the links, ports and map components are all real components in the preferable framework of the present invention.

Figs. 11-14 in Swamy show only scalar components (functoids). There is nothing to suggest that functoids can be assembled from other functoids. While Swamy does have a compiler to compile the map into a set of instructions for an engine, it does not produce a dataflow application. The Swamy invention does not execute dataflow applications.

The final Office Action cites Colliat for teaching multithreaded parallel transformation (final Office Action, pp. 10-11). Colliat describes two methods (col. 2), both based on standard practice for ETL (extract/transform/load) systems. In the first, you subdivide the data and process each of the pieces in parallel. In the second, the extraction, transformation and load stages are pipelined. Colliat prefers the second approach and executes the three stages of the process concurrently on three separate threads. However, there is no suggestion that dividing data and processes and pipelining in parallel would be applicable to Swamy, much less suggest the claim limitations recited in claims 17-20.

As noted above, Swamy does not produce a dataflow application. Therefore, modifying Swamy for parallel pipelining as taught by Colliat does not teach or suggest claims 17-20. Therefore, applying the tests of Graham and KSR, even if Swamy is combined with Colliat, there are substantial differences from the present claims and the proposed combination.

For at least the aforementioned reasons, Applicants believe claims 17-20 are patentably distinct over the combination of Swamy and Colliat.

* * *

For the foregoing reasons, it is submitted that the Examiner's rejection of pending claims 1-20 was erroneous, and reversal of the Examiner's decision is respectfully requested.

Respectfully Submitted,

/Charles D. Huston/
Charles D. Huston
Registration No. 31,027
Attorney for Appellant

Customer No. 35617
Date: December 8, 2008

VIII. APPENDIX

The present claims on appeal are as follows.

1. A method for developing a dataflow application comprising:

developing one or more data transformations using a host language;

assembling several data transformations having ports into a map component with links between ports using a declarative language for static assemblage and a host language for dynamic assemblage, wherein a map component performs a respective data transformation;

compiling one or more map components with syntactic and semantic analysis; and

synthesizing the compiled map components into an executable dataflow application including removing the design time links between ports.

2. The method of claim 1, wherein the steps of creating an executable dataflow application comprises:

synthesizing a hierarchical map component;

flattening the hierarchical map component; and

determining the executable dataflow application using the flattened hierarchical map component in conjunction with runtime properties.

3. The method of claim 1, further comprising encrypting the dataflow graphs prior to the step of compiling the map component.

4. The method of claim 1, wherein the host language for data transformation logic is an object-oriented programming language.

5. The method of claim 1, some of the map components implementing dynamic assemblage logic implemented in an object-oriented programming language.

6. The method of claim 1, some of the map components comprising a plurality of other map components arranged hierarchically.

7. The method of claim 1, some of the map components being static which consistently generate the same hierarchical map.

8. The method of claim 1, some of the map components being dynamic to generate different hierarchical maps dependent on properties and dynamic logic.

9. The method of claim 8, the dynamic map components receiving information concerning properties, port types, and dataflow graph implementation and configuring its properties and internal dataflow graph implementation based on said received information.

10. The method of claim 1, one or more of the map components having interface and implementation properties.

11. The method of claim 1, some of the map components ports configured for sending data and some configured for receiving data.

12. The method of claim 11, some of the ports being typed based on the type of data conveyed.

13. The method of claim 11, some of the ports being composite, comprising a plurality of hierarchical ports.

14. A computer-readable medium having a dataflow development system comprising:

a library of components, some of the components being scalar and comprising a data transformation, some of the components being composite, and comprising a hierarchy of other components representing data transformations;

a compiler to develop and assemble components into an executable dataflow application linking components subject to schema and properties constraints;

an executor which executes the dataflow application in parallel; and

tools for accessing the functionality of the compiler, executor and component library.

15. The system of claim 14, some of the components being encrypted and comprising a proprietary data transformation.

16. The system of claim 14, some of the components including ports for accepting and producing data whereby the map components are linked.

17. A computer-readable medium having a dataflow application development environment where map components are selected from a plurality of reusable map components each representing one or more data transformations, the selected map components are visually assembled into a dataflow application, patterns of parallelism recognized, and at least some of the components executed in parallel by assigning a number of threads to a respective number of data transformations, and at least some of the components are scalar components with each scalar component executed on a separate thread.

18. The environment of claim 17, wherein the map components define properties affecting design behavior.

19. The environment of claim 17, wherein the map components define properties affecting execution behavior.

20. The environment of claim 17, where the map components are assembled using ports for conveying data and declared as link points to other map components.

IX. EVIDENCE APPENDIX

No appeals, interferences, or judicial proceedings are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.